Storage-Efficient Data Prefetching for High Performance Computing

Yong Chen [1] Huaiyu Zhu [2] Hui Jin [3] Xian-He Sun [3]

[1] Department of Computer Science, Texas Tech University
{yong.chen@ttu.edu}
[2] Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
{hzhu10@illinois.edu}
[3] Department of Computer Science, Illinois Institute of Technology
{hjin6, sun}@iit.edu

Abstract. Data prefetching is widely adopted in modern high performance processors to bridge the ever-increasing performance gap between processor and memory. Many prefetching techniques have been proposed to exploit patterns among data access history that is stored in on-chip hardware table. We demonstrate that the table size has considerable impact on the performance of data prefetching. While a small table size limits the effectiveness of the prediction due to inadequate history, a large table is ex- pensive to be implemented on-chip and has longer latency. It is critical to find a storage-efficient data prefetching mechanism. We propose a novel Dynamic Signature Method (DSM) that stores the addresses efficiently to reduce the demand of storage for prefetching. We have carried out extensive simulation testing with a trace-driven simulator, CMP$im, and SPEC CPU2006 benchmarks. Experimental results show that the new DSM based prefetcher achieved better performance improvement for over half benchmarks compared to the existing prefetching approaches with the same storage consumption.

1 INTRODUCTION

The rapid advance of semiconductor technology drives the microprocessor performance and computing capability growing fast and steadily. However, the memory performance improvement remains lagging far behind the computational performance improvement. Multiple memory hierarchies have been served as the primary solution to mitigating the problem. However, due to the limited cache capacity and highly associative structure, large amount of off-chip accesses still spike the performance severely. Even worse, with the emerged multi-core architecture, the swift growth of the number of processor cores on chip puts more pressure than ever on the sluggish memory system. The cache capacity and bandwidth have not scaled up linearly with the number of cores, and the average off-chip accesses per core have also increased constantly. Unfortunately, this trend is predicted to continue in the next decade. There

adfa, p. 1, 2012.

is a great research need in investigating intelligent solution to boosting memory performance, such as data prefetching.

Data prefetcher has been widely considered as an effective technique to bridge the processor-memory performance gap. The basic principle of data prefetching is to explore the memory access patterns and predict the future requested memory access addresses speculatively before they are demanded by the processor. Correlation prefetching was proposed to detect repeated access patterns and make prefetches, and is accepted by many as a general and effective prefetching mechanism. The recurring data access addresses are usually kept in a hardware table, which is critical to a prefetcher for detecting patterns and issuing prefetches. The size of the tables can affect the prefetcher's performance significantly by either limiting the access pattern identification or prefetching lookahead distance. A large table is capable of providing more history lookup information to enhance the prefetching effectiveness. However, a large-size table consumes more precious chip space. Furthermore, a large prefetching table has long access latency, which limits the prefetching timeliness and the prefetcher's performance.

A storage-efficient data prefetching mechanism is desired, but remains a challenge in the research community. The goal of this study is to address this challenge that inherently limits the overall performance of prefetchers. We propose an innovative *Dynamic Signature Method (DSM)* to intelligently avoid the waste of table space and improve the storage efficiency for prefetchers. We incorporate this new idea with a well-known PC/DC (Program Counter localized Delta Correlation) prefetcher [12] and propose a new *Dynamic Signature-based Delta-correlation (DSD)* prefetcher. This DSD prefetcher successfully uses the signature (a partial address) instead of a full address whenever is possible to save the storage. It can be easily implemented by extending the Global History Buffer (GHB) [12], which is recognized as an efficient and effective structure for various prefetching schemes.

The rest of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 describes the methodology of the proposed dynamic signature method and introduces the design of the DSD prefetcher using this method. The evaluation settings and experimental results are discussed in Section 4. Section 5 reviews related works and compares with our work. Section 6 concludes this study.

2 BACKGROUND AND MOTIVATION

To achieve high prefetching accuracy, a stream localization technique was proposed and widely used for prefetcher design [2][11][12][16][17][20]. The rationale of this technique is that the global cache miss address stream (a full history of miss addresses) is separated into small groups according to a specific characteristic. For instance, the most common localization criteria used in data prefetching is the load/store instruction address (i.e. program counter, or PC) [2][12], which renders higher visibility of the access patterns and correlations within its sequence. Fig. 1 demonstrates an example of PC-localization technique, in which the global stream shown in Fig. 1(a) is localized to local streams shown in Fig. 1(b) according to PC values. Stream locali-

zation is beneficial in identifying access pattern efficiently and overcoming the dif-

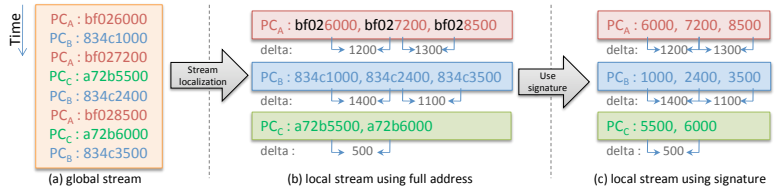

Fig. 1. Stream localization and signature method.

ficulty of processing complex data access patterns.

To support stream localization and prefetching based on it, one or more hardware tables, such as Global History Buffer (GHB) [12], are required to record useful data access history, discover the patterns, and generate prefetches. The capacity of the prefetching tables influences the performance considerably. Our proposed Dynamic Signature Method aims to alleviate the suffering caused by expensive hardware tables. This approach is motivated by the observation that once the global stream is localized, the spatial locality inside a local stream becomes much stronger than before, which indicates a series of data accesses fall in a small memory region for a specific local stream. Fig. 1(b) shows local streams after localization based on PC. From the figure we can see that the memory-access ad- dresses issued with the same PC have identical high-order bits (16 bits in stream PCA). This pattern can be observed in many applications. Once the high-order bits of an address are the same with its neighbor's, it is not necessary to store a full address in the prefetching table anymore. A straightforward solution to this problem is to store partial address (we term it as *signature*) instead of full address in the table. An example is shown in Fig. 1(c). This optimization methodology works effectively especially for those prefetching algorithms using deltas between adjacent addresses instead of addresses themselves.

3 DESIGN AND METHODOLOGY

3.1 Dynamic Signature Method

The principle of the dynamic signature method is that the prefetcher keeps collecting feedbacks derived from memory access footprints and adaptively decides what to store (partial or full address) in the table. The proposed DSM approach needs to deal with two challenges properly. One challenge is what signature should be stored (different forms of addresses), and the other is when it should be used (dynamic mechanism). In theory, various lengths of signatures can be used simultaneously. For example, a prefetcher can use four forms of signature with 4 bits, 8 bits, 16 bits and 32 bits (full address) respectively and uses one table entry for storing a 4-bit signature and 4 entries for a 32-bit full address. Supporting more forms of signatures simultaneously provides more choices for prefetcher and is helpful to maximize the storage efficiency, yet, produces more sophisticated table organization and intricate operation logic, which might consume more power and yield longer table access latency. Thus, appropriate signatures are crucial to enable DSM effectively. An adaptive mechanism is

also required for DSM since different signatures might be stored in the table, and the prefetcher needs to know which signature should be used to update the table. Next, we introduce a Dynamic Signature-based Delta- correlation (DSD) prefetcher to illustrate the challenges mentioned above and how they are addressed.

3.2 Dynamic Signature-Based Delta-Correlation Prefetching

Global History Buffer (GHB) is a FIFO table that is implemented as a circular buffer. Each GHB entry stores a miss address and a pointer which links the related GHB entries into an address list with chronological order. In the proposed DSD prefetcher, we enhance the GHB with minor modifications so that our signature method can utilize the table efficiently.

When the table needs to be updated, DSD prefetcher stores either a 10-bit signature or a 26-bit full address (6 bits of block line offset are excluded due to block-granularity prefetching) in the GHB. As shown in Fig. 2, one signature is stored in one GHB entry while a full address occupies two consecutive entries (or can be considered as a 2-wide entry) with the first entry holding the pointer and lower 10 bits of address, and the second entry immediately following the first one holding higher 16 bits of address. The index table is enhanced with a new field recording a hashed form of the latest address' high 16 bits for that local stream for feedback collection. The feedback scheme is part of the dynamic method and is discussed in the next subsection. The advantage of using only two forms of addresses is that the operation of this adaptive method can be simple and effective enough for performance improvement. In the meantime, 10-bit signature is selected based on its best average performance tested.

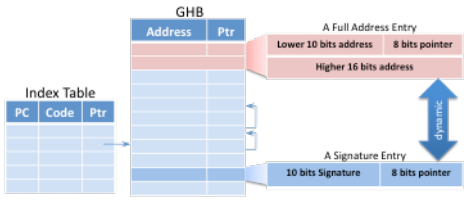

Fig. 2. Structure of DSD prefetcher.

4 EVALUATION AND ANALYSIS

4.1 Experimental Setup

4.1.1 Simulation Environment

In this study, a trace-driven simulator, CMP$im, that characterizes the memory system was adopted to evaluate the performance of prefetchers. The Data Prefetching Competition (DPC) committee [1] released a prefetcher kit that provides partial interface to make it feasible to integrate with an add-on prefetching module. The prefetcher kit contains Pin tool [10] and CMP$im simulator [7] to generate traces and conduct simulation. We utilize these features to evaluate the proposed dynamic signature method and the DSD prefetchers' performance. As shown in Table 1, the simulator is configured as an out-of-order processor with a 15-stage, 4-wide pipeline (maximum of two loads and maximum of one store can be issued every cycle) and perfect branch prediction. The cache follows LRU replacement policy.

The simulation testing is conducted with 18 benchmarks from SPEC-CPU2006 suite [18]. Several benchmarks in the set are omitted because of the compatibility issue or limited potential for improving the performance with prefetchers. The benchmarks are compiled using GCC 4.1.2 with –O3 optimization. We collect traces for all benchmarks by fast forwarding 40 billion instructions then running 500 million instructions and with the *ref* input size for all selected benchmarks.

Table 1. Architectural configuration

Window Size	128-entry
Issue Width	4
L1 Cache	32KB, 8-way
L2 Cache	512KB/1MB/2MB 16-way
Block Size	64 Bytes
L2 Cache Latency	20 cycles
Memory Latency	200 cycles
L2 Bandwidth	1 cycle/access
Memory Bandwidth	10 cycles/access
L2 MSHRs	32 entries
Prefetching Degree	8
DSD table size	1/2/4/8KB
PC/DC table size	1/2/4/8KB
G/DC table size	1/2/4/8KB
PC/CS table size	1/2/4/8KB

4.1.2 Prefetcher Configuration

The DSD prefetcher is designed based on PC/DC prefetcher, therefore, the PC/DC prefetcher is chosen for comparison. In addition, we compare the DSD prefetcher with another two widely-used variants of GHB prefetchers: Global-based Delta-Correlation (G/DC) [11] and PC- based Constant-Stride (PC/CS) [2][12] prefetchers. The storage budgets for each prefetcher are also shown in Table 1.

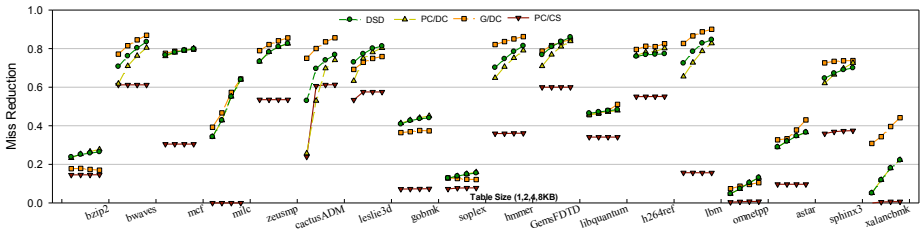

Fig. 3. L2 Miss Reduction for PC/DC, DSD, PC/CS and G/DC prefetchers with 1 MB LLC.

4.2 Performance Analysis

4.2.1 Miss Reduction

Fig. 3 reports the percentage of L2 cache misses reduced by PC/DC, DSD, G/DC and PC/CS prefetcher respectively. The PC/DC and DSD prefetchers adopt two deltas as context which is recommended in [12]. We vary the prefetching table size from 1KB to 8KB. The result shows that a large table size is helpful in reducing cache misses for most tested applications with DSD, PC/DC and G/DC prefetchers. However, the miss ratio of PC/CS prefetcher is not very sensitive to prefetcher table size because the constant stride algorithm requires relatively short historical miss infomation which is easily satisfied by a small table size. The DSD prefetcher reduces more L2 cache misses than the PC/DC prefetcher in 8 out of 18 benchmarks no matter what table size is used. In addition, the DSD prefetcher performs better under tight hardware budget.

With 1KB prefetcher table, DSD prefetcher outperforms PC/DC in 10 benchmarks and reaches the best improvement compared to PC/DC prefetcher in cactusADM (reduced 27% more misses). On average, the G/DC prefetcher is slightly better than others in reducing cache misses because it has high coverage. However, its poor accuracy limits the overall IPC improvement.

4.2.2 IPC Improvement

Fig. 4 shows the IPC (Instructions per Cycle) improvement with respect to the base case (performance without prefetching) of PC/DC, DSD, G/DC and PC/CS prefetchers. In this experiment, we have allocated 1KB, 2KB, 4KB and 8KB storage for all prefetching tables. Prefetching tables with 1~4KB are commonly used in most prefetching studies. We also include a relatively large table size of 8KB for performance testing. The simulation results show that DSD prefetcher significantly reduces the memory access latency and improves the IPC considerably. It outperforms PC/DC prefetcher in 11 out of 18 benchmarks - by as much as 19% in *leslie3d* - and reaches the peak improvements of 89.1% in *libquantum* under 1KB storage budget. In the meantime, with only 50% storage budget of the PC/DC prefetcher, the DSD prefetcher achieved the same performance as that of existing prefetchers for 1/3 tested benchmarks. Moreover, the DSD prefetcher outperforms PC/DC prefetcher regardless of table size in 8 out of 18 benchmarks. Note that the differences between DSD and PC/DC prefetcher become smaller when using 8KB tables because the strong demands for storage are relieved by large table size. Since applications tend to be more and more data intensive, a number of future applications might exhibit longer and more complex data access patterns which eventually require more storage to guarantee the effectiveness.

However, largely increasing the table size is impractical due to limited chip area and long access latency, which makes our study promising in saving the storage. In addition, Fig. 4 shows that the DSD prefetcher only underperforms PC/DC in few cases with the worst case of 1.7% performance drop in *mcf*, which is negligible and implies a strong reliability of DSD prefetcher. Recall that the signature-only method leads to significant performance loss in certain cases, whereas the results in Fig. 4 show our dynamic signature method is successful in eliminating this negative impact. Although the performance results differ in various benchmarks, the PC/DC algorithm beats the constant-stride and global delta algorithms in most cases.

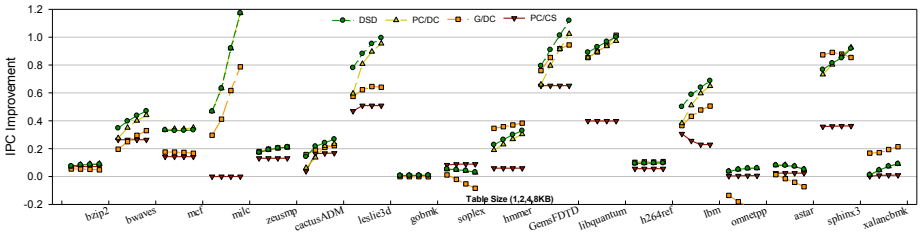

Fig. 4. IPC improvement for PC/DC, DSD, PC/CS and G/DC prefetchers with 1MB LLC.

5 RELATED WORK

Data prefetching has been extensively studied in past decades and widely used in commercial processors [4][15]. Sequential prefetching [3] is a simple mechanism that takes advantage of spatial locality and assumes the applications usually request next consecutive memory blocks in the near future. Stride prefetching [2] is widely used [4] due to its simplicity and effectiveness. It attempts to detect the constant stride patterns within PC-localized miss streams. A well-known address correlation prefetching using Markov chains was proposed in [8], which exploits the correlation between miss addresses and prefetch data based on a state transition diagram. Instead of memory address, the difference between addresses called deltas are used for correlation prefetching that was originally proposed for TLB [9] and then was used as PC/DC prefetching in data cache [12]. Along with PC/DC prefetcher, a prefetching storage structure known as the Global History Buffer (GHB) was proposed [12] with which various prefetching algorithms can be implemented. A comprehensive study on quantitative comparison of different hardware data cache optimizations has proved that the GHB based prefetcher achieves higher performance than others. To improve the performance and reduce the side effects, some prefetch controlling mechanisms are proposed such as Feedback-Directed prefetching (FDP) [19] and Hierarchical Prefetcher Aggressiveness Control [5].

To obtain high prefetching accuracy, the global miss stream fed to the prefetcher is usually localized in terms of some criterions. Stream localization by PC is the most widely used one [2][12]. AC/DC prefetching [11] uses GHB and spatial localization to implement delta correlation prefetching. Similarly, spatial memory streaming was proposed [16] to identify data correlations for commercial workload. Different from PC and spatial localizations, temporal localization proposed in [20] groups the addresses occurring in the same time period. A recent work called spatial-temporal streaming [17] that combines spatial and temporal streaming outperforms them.

Large size table is usually necessary to gain high performance. However, it consumes precious chip area and might also involve poor timeliness problem due to long table access latency. We have proposed stream timing mechanism recently [21] to improve the prefetching timeliness. To reduce the storage consumption, storing part of address in the table like what has been done in [4] is a possible solution. However, the effectiveness of this method is limited by the complexity of access patterns. In addition, the complex access patterns can lead to performance slowdown due to false predictions. Another category of prefetchers inherit from context-based value predictors [6][14] that encode their context in order to reduce the table size. Unfortunately, frequent hash collision limits the prefetching accuracy for those prefetchers. For large working sets in commercial applications, considerable table size is required to retain effectiveness.

6 CONCLUSION

In this study, we advance the state-of-the-art of data prefetching technique via introducing a novel dynamic signature method (DSM) aiming to reduce the demand of

precious on-chip storage for prefetchers. The proposed DSM takes advantage of the strong locality in the localized miss stream, thereby correlating the signatures instead of full addresses and enabling storage-efficient prefetching. We have extended the GHB-based PC/DC prefetcher with DSM and proposed a new Dynamic Signature-based Delta-correlation (DSD) prefetcher. We have carried out comprehensive simulation testing, and the result shows that the DSD prefetcher outperforms the PC/DC prefetcher in half of the benchmarks and underperforms it in very few cases. Moreover, compared with other GHB-based prefetchers, the DSD provides the highest average performance.

7 REFERENCES

1. DPC Homepage. *http://www.jilp.org/dpc*, 2008.
2. T.-F. Chen and J.-L. Baer. Effective hardware based data prefetching for high performance processors. IEEE Trans. Computers, 1995.
3. F. Dahlgren, M. Dubois, and P. Stenstrom. Fixed and adaptive sequential prefetching in shared memory multiprocessors. In *ICPP*, 1993.
4. J. Doweck. Inside Intel Core microarchitecture and smart memory access, Intel, 2006.
5. E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt. Coordinated control of multiple prefetchers in multi-core systems. In *MICRO*, 2009.
6. B. Goeman, H. Vandierendonck, and K. D. Bosschere. Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In *HPCA*, 2001.
7. A. Jaleel, R. S. Cohn, C.-K. Luk, and B. Jacob. CMP$im: a pin-based on- the-fly multi-core cache simulator. In *4th Workshop on Modeling, Benchmarking and Simulation*, 2008.
8. D. Joseph and D. Grunwald. Prefetching using Markov predictors. In *ISCA*, 1997.
9. G. B. Kandiraju and A. Sivasubramaniam. Going the distance for TLB prefetching: An application-driven study. In *ISCA*, 2002.
10. C.-K. Luk, R. S. Cohn, R. Muth, and et. al. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.
11. K. J. Nesbit, A. S. Dhodapkar, and J. E. Smith. AC/DC: an adaptive data cache prefetcher. In *PACT*, 2004.
12. K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *HPCA*, 2004.
13. D. G. Perez, G. Mouchard, and O. Temam. Microlib: A case for the quantitative comparison of micro-architecture mechanisms. In *MICRO*, 2004.
14. Y. Sazeides and J. E. Smith. The predictability of data values. In *MICRO*, 1997.
15. B. Sinharoy, R. N. Kalla, J. M. Tendler, and R. J. Eickemeyer. POWER5 system microarchitecture. *IBM Journal of Research and Development*, 2005.
16. S. Somogyi, T. F. Wenisch, et al. Spatial memory streaming. In *ISCA*, 2006.
17. S. Somogyi, T. F. Wenisch, M. Ferdman, and B. Falsafi. Spatio-temporal memory streaming. In *ISCA*, 2009.
18. C. D. Spradling. SPEC CPU2006 benchmark tools. *ACM SIGARCH Computer Architecture News*, 2007.
19. S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt. Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers. In *HPCA*, 2007.
20. T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Temporal streaming of shared memory. In *ISCA*, 2005.
21. H. Zhu, Y. Chen and X. Sun. Timing local streams: improving timeliness in data prefetching. In *ICS*, 2010.